Detailed (Topics 1–4)

1. Introduction to Requirements Engineering

◈ Definition:

Requirements Engineering (RE) is the **systematic discipline** of identifying, analyzing, documenting, validating, and managing the requirements of a software or system.

It is the bridge between **customer's needs** and the **developer's solution**.

◈ Objectives of RE:

1. **Identify needs** → Understand what stakeholders want.
2. **Ensure clarity** → Remove ambiguities and conflicts.
3. **Provide foundation** → Serve as base for design, coding, and testing.
4. **Minimize risk** → Reduce cost/time overruns by managing changes early.
5. **Stakeholder communication** → Improve communication between developers, users, managers.

◈ Importance in Software Development:

- Around **40–50% of project failures** occur due to poor requirement engineering.
- Correct RE leads to:
  - ✅ Less rework
  - ✅ Higher customer satisfaction
  - ✅ Better project planning

◈ Activities in Requirements Engineering:

1. **Requirement Elicitation** – Gathering requirements from stakeholders.
2. **Requirement Analysis** – Checking feasibility, resolving conflicts.

3. **Requirement Specification** – Writing requirements in a formal document (SRS).
4. **Requirement Validation** – Ensuring requirements are correct and agreed upon.
5. **Requirement Management** – Handling evolving requirements during project.

◈ Example (Banking App):

- Users want → Quick login, easy transfer.
- Bank wants → Security, regulatory compliance.
- Developers want → Feasible design within budget.
- RE ensures all perspectives are balanced.

◈ Possible Exam Questions:

- Define requirements engineering. Why is it important?
- Explain activities of requirements engineering with an example.

---

2. Software Requirements

◈ Definition:

Software requirements describe **what a software system should do** (functional) and **how it should perform** (non-functional).

They act as a **contract** between the client and the developer.

◈ Types of Software Requirements:

1. **Functional Requirements (FR):**
   - Describe **services and behaviors** the system should provide.
   - Example: "The system shall allow users to register with an email and password."
2. **Non-Functional Requirements (NFR):**
   - Describe **quality attributes** or constraints of the system.
   - Example: "The system should handle 1000 transactions per second."
3. **Domain Requirements:**
   - Come from the environment/domain in which the system will operate.

    - Example: “A hospital system must follow HIPAA compliance.”
4. **Process Requirements:**
    - Constraints on the development process.
    - Example: “System must be developed using Agile methodology.”

◈ Levels of Requirements:

- **User Requirements:** Written in natural language for end-users.
- **System Requirements:** Technical and detailed requirements for engineers.

◈ Characteristics of Good Requirements:

- Correct, complete, unambiguous, consistent, testable, traceable, feasible.

◈ Example (E-commerce website):

- FR → “User can add items to cart.”
- NFR → “System should load any page in <2 seconds.”
- Process → “System must be built using Java + Spring Boot.”

◈ Possible Exam Questions:

- Differentiate between functional and non-functional requirements with examples.
- Explain levels of requirements with a diagram.

---

3. Classification of Requirements

◈ Classification Based On:

1. **Functional vs Non-Functional**
    - FR: Defines specific functionality.
    - NFR: Defines quality or performance.
2. **Product vs Process**

    - Product → Directly describe the system (features, performance).
    - Process → Related to development (tools, methodology, standards).
3. **User vs System Requirements**
    - User → General needs written for non-technical audience.
    - System → Precise technical details for developers.
4. **Domain Requirements**
    - Based on application domain. Example: A medical system must store patient records securely for 10 years.

◈ Why Classification is Needed?

- Helps in **organizing requirements**.
- Makes it easier for **different stakeholders** to understand relevant requirements.
- Avoids missing any critical requirement.

◈ Example (University Management System):

- FR → "Student can register online."
- NFR → "System must be available 24/7."
- Domain → "System must comply with higher education standards."
- Process → "Development must follow ISO 9001."

◈ Exam Questions:

- Explain classification of requirements with suitable examples.
- Differentiate between user and system requirements.

---

4. Requirements Process

◈ Definition:

Requirements Process is the **step-by-step approach** followed to gather, analyze, document, validate, and manage requirements.

◈ Phases of Requirements Process:

1. **Requirement Elicitation:**
   - Collecting requirements using interviews, questionnaires, workshops, observations, prototyping.
2. **Requirement Analysis:**
   - Checking feasibility, resolving conflicts, prioritizing requirements.
3. **Requirement Specification:**
   - Writing requirements in a structured document → **Software Requirement Specification (SRS)**.
4. **Requirement Validation:**
   - Ensuring requirements are correct, complete, consistent → techniques: reviews, prototyping, testing.
5. **Requirement Management:**
   - Handling changes, traceability, version control, prioritization.

◈ Features of a Good Process:

- Iterative and incremental.
- Involves all stakeholders.
- Handles changes effectively.

◈ Example (Online Exam System):

- Elicitation → Students need online tests, Teachers want question banks.
- Analysis → Resolve conflicts (security vs ease of use).
- Specification → Create SRS.
- Validation → Check with teachers and students.
- Management → Handle future changes (new grading rules).

Detailed (Topics 5–8)

---

5. Levels / Layers of Requirements

◈ Definition:

Requirements are described at **different levels/layers** depending on who the audience is and how much detail is needed. Each level refines the previous one.

◈ Levels of Requirements:

1. **Business Requirements:**
   - High-level goals of the organization or customer.
   - Expresses **"Why" the system is needed.**
   - Example: "Increase online sales by 20% in 1 year."
2. **User Requirements:**
   - Describe **tasks and goals** from the end-user perspective.
   - Written in natural language (simple, non-technical).
   - Example: "Users should be able to track their orders."
3. **System Requirements:**
   - Detailed description of the system's **functionalities and constraints**.
   - Written in a technical format → SRS document.
   - Example: "System shall generate order tracking IDs and update status in real-time."
4. **Software Requirements:**
   - Focus specifically on **software functionalities**.
   - Example: "The software shall implement AES-256 encryption for payment data."
5. **Component / Design Requirements:**
   - Breaks system/software into modules and components.
   - Example: "Payment module should validate card using XYZ API."

◈ Importance of Levels:

- Clear communication with stakeholders.
- Avoids confusion between **"what" and "how"**.
- Ensures traceability from business goal → system design.

◈ Example (Online Food Delivery App):

- **Business:** "Improve customer convenience and increase sales."
- **User:** "Users can place orders via mobile app."
- **System:** "System shall store orders in central database."
- **Software:** "Mobile app shall be built in React Native."
- **Component:** "Database must have an orders table with fields X, Y, Z."

◈ Exam Questions:

- Explain different levels of requirements with suitable examples.
- Why is requirement layering important in software engineering?

---

6. Requirement Characteristics

◈ Definition:

A **good requirement** must have certain characteristics to avoid ambiguity, inconsistency, and misinterpretation.

◈ Characteristics of Good Requirements (IEEE Standard):

1. **Correct** – Accurately represents stakeholder needs.
2. **Complete** – All important requirements are included.
3. **Unambiguous** – Written clearly with no double meaning.
4. **Consistent** – Should not conflict with other requirements.
5. **Feasible** – Must be technically and financially possible.
6. **Verifiable (Testable)** – Can be tested during validation.
7. **Traceable** – Can be linked back to business goals.
8. **Modifiable** – Easy to change/update.
9. **Prioritized** – Must indicate level of importance (High/Medium/Low).

◈ Why Requirement Characteristics Are Important?

- Prevent miscommunication.

- Reduce cost of rework.
- Helps in testing and validation.
- Ensures stakeholder satisfaction.

◈ Example (ATM System):

❌ Bad Requirement: "ATM should work fast." (Ambiguous, not testable)
✅ Good Requirement: "ATM shall dispense cash within 10 seconds." (Clear, measurable, testable)

◈ Exam Questions:

- List and explain characteristics of good requirements with examples.
- Why is testability important in requirement writing?

---

7. Analyzing Quality Requirements

◈ Definition:

Quality Requirements = **Non-Functional Requirements (NFRs)** such as performance, reliability, security, usability, maintainability, etc.

Analyzing quality requirements means ensuring that these NFRs are **clear, measurable, and achievable**.

◈ Steps in Analyzing Quality Requirements:

1. **Identify Quality Attributes**
   - Performance, scalability, security, usability, portability, reliability.
2. **Quantify Requirements**
   - Example: "System must handle 1000 concurrent users with <2s response time."
3. **Check Feasibility**
   - Can the current technology achieve this performance/security?
4. **Prioritize**
   - Example: In a banking system, security > usability.

5. **Trade-off Analysis**
    - High performance may reduce maintainability. Balance is required.

◈ Common Quality Attributes:

- **Performance** → Response time, throughput.
- **Security** → Authentication, encryption.
- **Reliability** → Availability, fault tolerance.
- **Usability** → User-friendly interface.
- **Maintainability** → Easy to fix/update.
- **Portability** → Can run on different platforms.

◈ Example (Online Exam System):

- NFR: "System must prevent cheating."
- Analysis: Add proctoring, secure login, monitor browser.
- Trade-off: High security may reduce usability for students.

---

8. Software Requirements in the Context of Systems Engineering

◈ Definition:

Systems Engineering (SE) = **Interdisciplinary approach** for designing and managing complex systems (hardware + software + processes).

Software requirements are **a subset of system requirements** and must align with the entire system design.

◈ Relationship Between Systems Engineering & Software Requirements:

1. **System Requirements → Software Requirements**

    - Systems engineering defines overall system needs.
    - Software engineers extract software-specific needs.
2. **Integration with Hardware Requirements**
    - Example: An autopilot system → software must interact with sensors (hardware).
3. **Traceability**
    - Software requirements must trace back to system-level goals.
4. **Interfaces**
    - Software must define clear interaction with hardware, users, and external systems.

◈ Example (Self-Driving Car):

- **System Requirement:** "Car must maintain lane automatically."
- **Software Requirement:** "Software must process camera images at 60 FPS to detect lane markings."
- **Hardware Requirement:** "Car must have HD camera and sensors."

◈ Why Context is Important?

- Avoids mismatch between software and hardware.
- Ensures the software meets overall system goals.
- Helps in safety-critical domains (aviation, healthcare, automotive).

(Topics 9–12)

---

9. Requirement Evolution

◈ Definition:

Requirement Evolution is the **continuous change and refinement of requirements** throughout the software/system life cycle.
Since business needs, technology, and environments change over time → requirements cannot remain static.

◈ Causes of Requirement Evolution:

1. **Changing Business Needs**
   - Market trends, policies, or regulations change.
   - Example: A bank must update software due to new government tax rules.
2. **Changing User Needs**
   - Users discover new expectations during usage.
   - Example: Students demand a mobile version of an e-learning platform.
3. **Technology Advancement**
   - New platforms, devices, or frameworks become available.
   - Example: Migrating from desktop-based system to cloud-based solution.
4. **Errors and Omissions**
   - Some requirements missed or misunderstood earlier.
5. **Environment Change**
   - Laws, competitors, or security threats force updates.
   - Example: GDPR law forced apps to update privacy features.

◈ Types of Requirement Evolution:

1. **Corrective Evolution** → Fixing mistakes in requirements.
2. **Adaptive Evolution** → Adjusting requirements to new environment/technology.
3. **Perfective Evolution** → Improving requirements to increase performance/usability.
4. **Preventive Evolution** → Adding requirements to avoid future problems.

◈ Process of Requirement Evolution:

1. Identify the need for change.
2. Document the change request.
3. Analyze impact of the change.
4. Approve or reject the change.
5. Update requirement document & inform stakeholders.
6. Maintain traceability for future audits.

◈ Example (E-Commerce App):

- Initial Requirement: “System shall support online payments.”

- Evolved Requirement: “System shall support multiple digital wallets (PayPal, Apple Pay, Google Pay) and QR code payments.”

◈ Challenges in Requirement Evolution:

- Scope creep (uncontrolled growth of requirements).
- High cost of frequent changes.
- Maintaining traceability.
- Resistance from stakeholders.

◈ Exam Questions:

- Define requirement evolution. What are its causes and challenges?
- Differentiate between corrective, adaptive, and perfective evolution.

---

10. Requirement Traceability

◈ Definition:

Requirement Traceability is the **ability to link each requirement throughout the software development life cycle (SDLC)** to ensure nothing is missed.

◈ Types of Traceability:

1. **Forward Traceability**
   - Tracks requirement → design → code → test cases.
   - Ensures every requirement is implemented.
   - Example: Business goal → user stories → system modules → test cases.
2. **Backward Traceability**
   - Tracks code → design → requirement.
   - Ensures no extra/unnecessary features are added.
3. **Bi-directional Traceability**
   - Combination of both forward and backward.
   - Most reliable approach in complex systems.

◈ Traceability Matrix (RTM):

A table mapping requirements to their implementation and testing.

Example:

Requirement ID	Design Component	Code Module	Test Case ID	Status
R-01	Login Page	Login.java	TC-101	Passed
R-02	Payment Gateway	Payment.java	TC-202	In Progress

◈ Importance of Traceability:

- Ensures no requirement is missed.
- Helps in impact analysis of requirement changes.
- Helps during audits (especially in safety-critical systems).
- Increases project transparency.

◈ Example (Hospital Management System):

Requirement: "System shall store patient medical history."

- Traceability → DB design table → Code module → Test case verifying data storage.

---

11. Requirement Prioritization

◈ Definition:

Requirement Prioritization is the process of **deciding which requirements should be implemented first**, based on importance, value, and constraints.

◈ Why Prioritization is Needed?

- Limited time, budget, and resources.
- All requirements are not equally important.
- Helps focus on delivering maximum value early.

◈ Factors Affecting Prioritization:

1. **Business Value** – Which requirements generate more profit?
2. **Stakeholder Needs** – Whose requirements are most critical?
3. **Risk** – High-risk requirements may need early implementation.
4. **Cost & Effort** – Easy, low-cost requirements may be implemented first.
5. **Dependencies** – Some requirements depend on others.

◈ Prioritization Techniques:

1. **MoSCoW Technique**
   - Must have, Should have, Could have, Won't have.
2. **100-Dollar Test**
   - Stakeholders distribute 100 points (or dollars) across requirements to show importance.
3. **Pairwise Comparison (AHP - Analytical Hierarchy Process)**
   - Compare requirements in pairs to see which is more important.
4. **Ranking**
   - Simple high, medium, low priority assignment.
5. **Kano Model**
   - Classifies features into **Basic (must-have), Performance (good to have), Excitement (delighters)**.

◈ Example (Food Delivery App):

- **Must Have:** Secure login, order placement, payment.
- **Should Have:** Order tracking.
- **Could Have:** AI-based recommendations.
- **Won't Have (for now):** Drone delivery.

◈ Exam Questions:

- Explain MoSCoW method of requirement prioritization with example.
- Why is requirement prioritization important in software development?

---

12. Trade-off Analysis

◈ Definition:

Trade-off analysis is the process of **balancing conflicting requirements** where satisfying one may negatively affect another.

◈ Why Trade-offs are Needed?

- Limited time, cost, and resources.
- Conflicts between performance vs. security, usability vs. functionality, cost vs. quality.

◈ Common Trade-off Scenarios:

1. **Performance vs. Security**
   - Strong encryption → slower performance.
2. **Usability vs. Functionality**
   - More features → system becomes complex.
3. **Cost vs. Quality**
   - Higher quality demands → more cost.
4. **Time vs. Features**
   - Deliver quickly → fewer features.

◈ Trade-off Analysis Process:

1. Identify conflicting requirements.
2. Gather stakeholder opinions.
3. Evaluate alternatives (cost-benefit analysis).
4. Select the most balanced solution.

5. Document decision for transparency.

◈ Example (Online Banking App):

- Requirement A: “High security with multi-factor authentication.”
- Requirement B: “Quick login for users.”
  👉 Trade-off: Introduce biometric login (face/fingerprint) → maintains security **and** fast access.

◈ Tools for Trade-off Analysis:

- Cost-Benefit Analysis (CBA).
- Decision Matrix.
- Risk-Impact Analysis.

(Extended Detailed Notes)

---

13. Risk Analysis and Impact Analysis

◈ Introduction

- Every software project faces **uncertainties**. These uncertainties can harm project objectives (time, cost, quality, or scope).
- **Risk Analysis** is about predicting these uncertainties (risks), estimating their effect, and preparing responses.
- **Impact Analysis** is about studying the effect of changes in requirements on the entire system.

◈ Risk Analysis in Detail

1. Definition

Risk analysis is the **systematic process of identifying, analyzing, and managing risks** that may affect software requirements or project success.

2. Why Risk Analysis is Important?

- Requirements often change → risks of scope creep.
- Early identification of risks saves cost/time.
- Prevents failure in large projects.

3. Types of Risks in Requirements Engineering

- **Requirement Risks**: Ambiguous, incomplete, or volatile requirements.
- **Technical Risks**: Technology may not support the requirement.
- **Operational Risks**: Users may resist adopting new software.
- **Business Risks**: Requirement may not provide expected ROI.
- **Regulatory/External Risks**: Law changes, competitor's actions, or environmental issues.

4. Risk Analysis Process

1. **Risk Identification** – Brainstorm, expert interviews, checklists.
2. **Risk Assessment** – Assign probability (High/Med/Low) and severity.
3. **Risk Prioritization** – Rank risks by "Risk Exposure = Probability × Impact".
4. **Risk Mitigation** – Prepare action plan.
    - Avoid (change plan to remove risk).
    - Reduce (apply controls).
    - Transfer (insurance, outsourcing).
    - Accept (live with it if cost of control > risk).
5. **Risk Monitoring** – Regular tracking during project lifecycle.

◈ Impact Analysis in Detail

1. Definition

Impact Analysis is the process of **analyzing the effect of a proposed change in a requirement** on the system.

2. Need for Impact Analysis

- Requirements are dynamic.
- Each change may affect design, code, testing, documentation.
- Avoids "ripple effects" of change.

3. Steps in Impact Analysis

1. Analyze change request.
2. Identify related requirements.
3. Check impact on design, modules, database, interfaces, and tests.
4. Estimate cost, time, and resources.
5. Decide whether to approve change.

4. Tools Used

- Requirement Traceability Matrix (RTM).
- Dependency Graphs.
- CASE tools.

◈ Example

- A banking app requirement: "Add biometric login".
- Impact: Affects **UI**, **database (new fields for fingerprint template)**, **security architecture**, and **testing**.
- Risk: High cost of implementation, possible legal issues in some regions.

◈ Expected Exam Question Answer

Q: What is Risk Analysis and Impact Analysis in Requirements Engineering? Explain with process and example.
Ans: Risk Analysis is … [then explain with 5 steps].
Impact Analysis is … [then explain with steps + example].

---

14. Requirement Management

◈ Introduction

Requirement management ensures that **all requirements are clearly documented, controlled, updated, and implemented correctly** throughout the project.

◈ Definition

Requirement management is the **continuous process of capturing, organizing, analyzing, and controlling requirements and their changes during the project lifecycle.**

◈ Objectives of Requirement Management

- Maintain consistency among requirements.
- Control requirement changes.
- Keep stakeholders aligned.
- Maintain **traceability** (from requirement → design → testing).
- Improve communication between developers and clients.

◈ Main Activities

1. **Requirement Identification** – Assign unique IDs.
2. **Change Management** – Process to evaluate and approve/reject changes.
3. **Version Control** – Maintain history of requirement evolution.
4. **Status Tracking** – Track requirement states (proposed, approved, implemented).
5. **Documentation** – Keep requirements clear and updated.
6. **Requirement Traceability** – Link requirements with design, code, and tests.

◈ Challenges in Requirement Management

- Frequent changes by stakeholders.
- Conflicts between different requirements.
- Large projects = thousands of requirements (hard to track).
- Cost of managing complex requirements.

◈ Example

In an **E-commerce project**, requirement: "Users should pay using PayPal."
Later, client requests: "Add cryptocurrency payment option."
👉 Requirement management ensures change request is analyzed, approved, documented, and implemented.

◈ Exam Question Answer

- Define requirement management.
- List its activities.
- Explain why it is important.

---

15. Interaction Between Requirement and Architecture

◈ Introduction

Requirements define **what the system should do**, and architecture defines **how it will be implemented**. Both are interdependent.

◈ Impact of Requirements on Architecture

- **Performance Requirement** → architecture must support load balancing, caching.
- **Security Requirement** → encryption layers, firewall in architecture.
- **Scalability Requirement** → microservices/cloud architecture.
- **Integration Requirement** → layered architecture with APIs.

◈ Impact of Architecture on Requirements

- Some requirements may not be feasible with chosen architecture.
- Example: If architecture is monolithic → hard to meet “high scalability” requirement.
- Example: If architecture is cloud-based → requirement of offline access may need change.

◈ Requirement–Architecture Trade-off

- Architecture restricts requirements (feasibility).
- Requirements shape architecture (design decisions).

◈ Example

Requirement: “System must handle 10,000 users per second.”
👉 Architecture: Requires distributed database, caching server, load balancers.

◈ Exam Question Answer

Q: Explain interaction between requirements and architecture with example.
Ans: Requirements define … Architecture influences … Example …

---

16. Requirement Elicitation

◈ Definition

Requirement elicitation is the **process of gathering requirements from stakeholders** using systematic techniques.

◈ Why Elicitation is Important?

- Stakeholders may not clearly state needs.
- Misunderstanding leads to project failure.
- Good elicitation = accurate requirements = successful project.

◈ Elicitation Process

1. **Identify stakeholders** – Users, clients, managers, developers.

2. **Understand domain** – Study business context.
3. **Collect requirements** – Use interviews, workshops, observations.
4. **Refine requirements** – Remove conflicts and ambiguities.
5. **Document requirements** – Use Software Requirement Specification (SRS).

◈ Challenges in Elicitation

- Stakeholders unclear about goals.
- Conflicting needs.
- Communication barriers.
- Changing needs over time.

◈ Techniques for Elicitation (brief overview)

- **Interviews**
- **Workshops / JAD (Joint Application Development)**
- **Questionnaires**
- **Observation**
- **Prototyping**
- **Document Analysis**

◈ Example

For an **Online Food Delivery App**:

- Customers want quick order → simple UI.
- Restaurants want menu management.
- Delivery riders want route optimization.
- Management wants analytics dashboard.

  👉 Elicitation combines all perspectives.

(Extended Detailed Notes)

---

17. Phased Integration

◈ Definition

Phased Integration is a **software integration strategy** in which different modules/components of a system are combined **step by step, in planned phases**, instead of integrating all at once.

◈ Key Objectives

- Reduce risk of integration failure.
- Detect errors early by testing after each phase.
- Manage complex systems systematically.

◈ Process of Phased Integration

1. **Module Development** – Individual components are designed & coded.
2. **Planning the Integration Sequence** – Decide which module should be integrated first.
3. **Incremental Integration** – Add modules step by step.
4. **Testing After Each Phase** – Verify correctness before moving to the next phase.
5. **Final System Integration** – After all phases, the complete system is tested.

◈ Advantages

- Errors detected early.
- Lower debugging cost compared to big-bang approach.
- Easier to manage project complexity.
- Better quality control.

◈ Disadvantages

- More time-consuming.
- Requires careful planning of integration sequence.
- Higher initial cost compared to “big bang” integration.

◈ Example

Suppose we are developing an **Online Shopping System**:

- **Phase 1:** Integrate User Login + Registration.
- **Phase 2:** Add Product Browsing.
- **Phase 3:** Add Shopping Cart.
- **Phase 4:** Add Payment System.
- **Phase 5:** Add Order Tracking.

Each phase is tested before moving to the next.

◈ Exam Question Answer

Q: Define Phased Integration with advantages, disadvantages, and example.
Ans: Phased integration is … (definition + process + example).

---

18. Incremental Integration

◈ Definition

Incremental Integration is a **testing and integration strategy** where software modules are integrated **one by one** and tested immediately after each addition.

◈ Objectives

- Ensure modules work together correctly.
- Isolate bugs quickly after each increment.
- Allow partial system delivery (usable product even before full system is ready).

◈ Process of Incremental Integration

1. Develop module A → Test.
2. Add module B to A → Test A+B.

3. Add module C to A+B → Test A+B+C.
   ... continue until full system is built.

◈ Strategies for Incremental Integration

1. **Top-Down Integration**
   - Start with top-level module (main controller).
   - Add lower modules step by step.
   - Use **stubs** (dummy code) for unavailable modules.
2. **Bottom-Up Integration**
   - Start with lowest-level modules.
   - Integrate upward until reaching top-level.
   - Use **drivers** (temporary code to call modules).
3. **Sandwich (Hybrid) Integration**
   - Combine both top-down and bottom-up approaches.

◈ Advantages

- Easier to locate and fix errors.
- Working system is available early.
- Testing is more systematic.

◈ Disadvantages

- Requires more planning & effort.
- Testing may take longer than big-bang.

◈ Example

In a **Banking System**:

- Step 1: Integrate Account Creation.
- Step 2: Add Balance Inquiry.
- Step 3: Add Deposit/Withdrawal.
- Step 4: Add Online Transactions.
- Step 5: Add Loan Management.

System is usable from Step 2 onwards.

◈ Exam Question Answer

Q: Explain Incremental Integration with types and example.
Ans: Incremental integration is … (definition → process → types → example).

---

19. Feature-Oriented Integration

◈ Definition

Feature-Oriented Integration is an **integration strategy** where modules are integrated based on **system features**, rather than their hierarchy or position in architecture.

◈ Key Objectives

- Deliver features that provide business value early.
- Allow customers to see working features quickly.
- Prioritize critical or high-value features first.

◈ Process of Feature-Oriented Integration

1. Identify **features** (functional units like “Login”, “Search”, “Checkout”).
2. Prioritize features (critical → optional).
3. Develop and integrate one feature at a time.
4. Test after each feature integration.
5. Deliver working features to stakeholders for feedback.

◈ Advantages

- User gets working features early.
- Easier to adapt to changing customer needs.
- Increases customer satisfaction.

- Supports **Agile Development**.

◈ Disadvantages

- Requires careful feature prioritization.
- Dependencies between features may complicate integration.
- High coordination needed among teams.

◈ Example

In a **Food Delivery App**:

- **Feature 1:** User Registration + Login.
- **Feature 2:** Restaurant Search.
- **Feature 3:** Order Placement.
- **Feature 4:** Online Payment.
- **Feature 5:** Order Tracking.

👉 Features are integrated and tested one by one.

◈ Exam Question Answer

Q: What is Feature-Oriented Integration? Explain with process and example.
Ans: Feature-oriented integration is … (definition + process + example).

---

20. Requirement Phase in SDLC

◈ Definition

The **Requirement Phase** in SDLC (Software Development Life Cycle) is the **first and most important phase**, where system requirements are gathered, analyzed, and documented in an SRS (Software Requirement Specification).

◈ Objectives of Requirement Phase

- Understand what the customer needs.
- Define system scope and boundaries.
- Document clear and testable requirements.
- Reduce risk of project failure.

◈ Activities in Requirement Phase

1. **Requirement Elicitation** – Collecting requirements (interviews, workshops, surveys).
2. **Requirement Analysis** – Checking consistency, completeness, feasibility.
3. **Requirement Documentation** – Writing SRS (Software Requirement Specification).
4. **Requirement Validation** – Reviewing with stakeholders to confirm correctness.

◈ Types of Requirements in SDLC

1. **Functional Requirements** – Define what the system should do.
    - Example: "System should allow users to transfer money."
2. **Non-Functional Requirements** – Define how the system should behave.
    - Example: "System must process transactions in <2 seconds."
3. **Business Requirements** – Define organizational goals.
4. **User Requirements** – Define user's needs in natural language.

◈ Importance of Requirement Phase

- Basis of design, coding, and testing.
- Reduces misunderstanding between clients and developers.
- Avoids costly rework in later stages.

◈ Challenges in Requirement Phase

- Unclear requirements from stakeholders.
- Changing requirements over time.
- Conflicts between stakeholders.

◈ Example

In a **Hospital Management System**:

- Functional Requirement: "Doctors can update patient reports."
- Non-Functional Requirement: "System should support 1000 users at once."

👉 These are documented in the Requirement Phase for design reference.

(Extended Detailed Notes)

---

21. Types of Graph Neural Networks (GNNs)

◈ Definition

Graph Neural Networks (GNNs) are **deep learning models** designed to work with **graph-structured data** (nodes + edges). Unlike CNNs (for images) or RNNs (for sequences), GNNs can directly model relationships in networks (social networks, molecules, recommendation systems).

◈ Types of GNNs

1. **Graph Convolutional Networks (GCNs)**
   - Extension of CNN for graph data.
   - Works by aggregating information from **neighboring nodes**.
   - Example: In a social network, GCN can predict a person's interests based on their friends.
2. **Graph Attention Networks (GATs)**
   - Use **attention mechanism** to give different importance (weights) to neighboring nodes.
   - More flexible than GCNs.
   - Example: In recommendation systems, a close friend's opinion is given more weight.
3. **Graph Autoencoders (GAEs)**
   - Used for **graph representation learning**.

    - Encode graph into a lower-dimensional vector, then reconstruct it.
    - Example: Useful for **link prediction** in networks.
4. **Graph Recurrent Neural Networks (GRNNs)**
    - Combine RNNs with graph structures.
    - Handle **sequential graph data** (time-evolving graphs).
    - Example: Predicting traffic in a road network over time.
5. **Spatial vs Spectral GNNs**
    - **Spatial GNNs**: Work by directly aggregating neighborhood information.
    - **Spectral GNNs**: Use spectral graph theory and Laplacian matrix to perform convolution.

◈ Advantages

- Capture complex relationships.
- Handle dynamic & irregular data structures.
- Effective for real-world networks.

◈ Disadvantages

- High computational cost.
- Scalability issues with very large graphs.

◈ Exam Question Answer

Q: Explain different types of Graph Neural Networks with examples.
Ans: The main types of GNNs are GCN, GAT, GAE, GRNN … (with definitions + examples).

---

22. Why Use Graph Neural Networks (GNNs)?

◈ Introduction

Graphs are everywhere: social media, biological networks, road networks, knowledge graphs. Traditional ML models **cannot directly process graph-structured data**. This is why we use GNNs.

◈ Reasons to Use GNNs

1. **Graph-Structured Data Representation**
   - GNNs naturally handle nodes & edges.
   - Example: Representing users & friendships on Facebook.
2. **Relational Learning**
   - GNNs learn from **relationships** between entities.
   - Example: Product recommendations (Amazon recommends based on related purchases).
3. **Scalability**
   - Work on large-scale, real-world graphs.
4. **Applications Across Domains**
   - Social Network Analysis
   - Traffic Prediction
   - Protein/Molecule Structure Prediction
   - Fraud Detection (Banking, Finance)
5. **Better Performance**
   - Outperforms traditional ML on graph data.

◈ Advantages

- Works on structured + unstructured data.
- Extracts hidden patterns from relationships.
- Useful in AI applications like **ChatGPT's knowledge graph connections**.

◈ Limitations

- Complex to train.
- Expensive for large dynamic graphs.

◈ Exam Question Answer

Q: Why are Graph Neural Networks used? Explain with examples.
Ans: GNNs are used because … (points with examples).

---

23. Advantages of Self-Supervised Learning (SSL)

◈ Definition

Self-Supervised Learning is a **machine learning paradigm** where the system generates labels from the raw data itself, without manual labeling. It lies **between supervised and unsupervised learning**.

◈ Advantages

1. **No Need for Labeled Data**
   - Reduces cost of manual data labeling.
   - Example: Training a model on YouTube videos without human annotations.
2. **Utilizes Large Amounts of Data**
   - Works on unlabeled datasets (which are abundant).
3. **Better Generalization**
   - Learns richer representations → performs well on multiple tasks.
4. **Transfer Learning**
   - Pre-trained SSL models can be fine-tuned for other tasks.
   - Example: BERT in NLP, trained with SSL and reused in many tasks.
5. **Robustness**
   - Learns from data context → resistant to noise.
6. **Scalability**
   - Easily scales to millions of samples.

◈ Applications

- NLP (BERT, GPT models).
- Computer Vision (image captioning, face recognition).
- Speech Recognition.

◈ Limitations

- Requires very large datasets + computational power.
- Pretext task (self-generated labels) may not always align with final task.

◈ Exam Question Answer

Q: Explain the advantages of Self-Supervised Learning.
Ans: SSL is … (definition + 6 advantages + applications).

---

24. What is AI (Artificial Intelligence)?

◈ Definition

Artificial Intelligence (AI) is the **branch of computer science** that aims to create systems capable of **thinking, learning, reasoning, and decision-making** like humans.

◈ Objectives of AI

- Automate human-like tasks.
- Improve problem-solving.
- Create systems that adapt and learn.

◈ Types of AI

1. **Weak AI (Narrow AI)**
    - Designed for specific tasks.
    - Example: Google Maps, Siri, Chatbots.
2. **Strong AI (General AI)**
    - Human-like intelligence.
    - Can perform any intellectual task.
    - Example: Still in research stage.
3. **Super AI**
    - Beyond human intelligence.
    - Future concept, can outperform humans in all areas.

◈ Applications of AI

- Healthcare (disease detection).
- Autonomous Cars.

- Robotics.
- Finance (fraud detection).
- Customer Service (chatbots).

◈ Advantages

- Increases efficiency.
- Reduces human error.
- Works 24/7.

◈ Disadvantages

- Expensive to build.
- Job displacement.
- Ethical concerns.

(Detailed Notes)

---

25. Deep Learning and Its Types

◈ Definition

Deep Learning is a **subset of Machine Learning** that uses **Artificial Neural Networks (ANNs) with multiple hidden layers** to automatically learn features and patterns from large datasets.

It is inspired by how the **human brain (neurons and synapses)** processes information.

◈ Key Features of Deep Learning

- Learns features automatically (no manual feature engineering).
- Requires large datasets and computational power (GPUs).
- Performs better with **unstructured data** like images, speech, and text.

◈ Types of Deep Learning Models

1. **Convolutional Neural Networks (CNNs)**
   - Specially designed for **image and video processing**.
   - Uses convolution filters to detect features like edges, shapes, objects.
   - Example: Face recognition, medical image analysis.
2. **Recurrent Neural Networks (RNNs)**
   - Designed for **sequential/time-series data**.
   - Have memory to remember past inputs.
   - Example: Stock price prediction, speech recognition.
3. **Long Short-Term Memory Networks (LSTMs)**
   - An advanced type of RNN.
   - Overcomes vanishing gradient problem by using gates (input, output, forget).
   - Example: Language translation, text generation.
4. **Generative Adversarial Networks (GANs)**
   - Two networks (Generator + Discriminator) compete.
   - Generates realistic synthetic data.
   - Example: Deepfake images, art generation.
5. **Deep Reinforcement Learning (DRL)**
   - Combines deep learning with reinforcement learning.
   - Example: Self-driving cars, AlphaGo (Google's AI beating human champions).

◈ Applications of Deep Learning

- Natural Language Processing (NLP).
- Computer Vision (object detection, face recognition).
- Healthcare (disease prediction).
- Autonomous Vehicles.
- Robotics.

◈ Exam Answer

Q: Define Deep Learning and explain its types with examples.
Ans: Deep learning is … (definition + types + examples).

---

26. Decision Tree and Random Forest

◈ Decision Tree

- A **supervised machine learning algorithm** used for classification & regression.
- Works like a **tree structure**:
  - **Root Node** → starting point (main feature).
  - **Decision Nodes** → internal nodes for conditions.
  - **Leaf Nodes** → final output/prediction.

Example:
If Weather = Sunny → Play = Yes
If Weather = Rainy → Play = No

◈ Advantages of Decision Tree

- Easy to understand & interpret.
- Works with numerical + categorical data.
- Handles non-linear relationships.

◈ Disadvantages of Decision Tree

- Overfitting (too complex).
- Sensitive to small changes in data.

◈ Random Forest

- An **ensemble learning method** based on multiple decision trees.
- Each tree is trained on a random subset of data → final prediction is **majority vote (classification)** or **average (regression)**.

Advantages:

- Reduces overfitting.
- More accurate than single decision tree.
- Works well on large datasets.

Example:

- Medical diagnosis.
- Fraud detection in banking.

◈ Exam Answer

Q: Differentiate between Decision Tree and Random Forest with examples.
Ans: Decision tree is … Random forest is … (add advantages).

---

27. Redundancy, Load Balancing, Cost Optimization, and Automated Process Management

◈ Redundancy

- Having **backup systems/components** in case the main one fails.
- Ensures **reliability and fault tolerance**.
- Example: Data stored in multiple servers (RAID systems).

◈ Load Balancing

- Distributes workload across multiple servers to ensure:
  - No single server is overloaded.
  - Faster response time.
  - High availability.
- Example: Cloud services (AWS, Google Cloud).

◈ Cost Optimization

- Reducing IT & cloud infrastructure costs while maintaining performance.
- Techniques:
  - Auto-scaling (use resources only when needed).
  - Using spot instances in cloud.
- Example: E-commerce website using cloud auto-scaling during sales.

◈ Automated Process Management (APM)

- Using **automation tools** to manage business processes.
- Reduces manual effort, increases efficiency.
- Example: Automated backup, DevOps pipelines.

◈ Exam Answer

Q: Explain Redundancy, Load Balancing, Cost Optimization, and Automated Process Management with examples.
Ans: Redundancy is … Load balancing is … (with real-world examples).

---

28. Strong AI vs Weak AI (Narrow AI vs General AI)

◈ Weak AI (Narrow AI)

- AI systems designed to perform **specific tasks**.
- They don't possess real intelligence, only mimic it.
- Examples:
    - Siri, Alexa.
    - Google Maps.
    - Spam Filters.

◈ Strong AI (General AI)

- AI that can perform **any intellectual task like a human**.
- Can **learn, reason, adapt, and make decisions independently**.
- Still in **research stage**, not yet achieved.
- Examples:
    - Hypothetical AI like in movies (Jarvis in Iron Man).
    - Advanced research in robotics & cognitive science.

🔷 Key Differences

Feature	Weak AI (Narrow)	Strong AI (General)
Scope	Task-specific	General-purpose
Intelligence Level	Limited, pre-programmed	Human-like
Examples	Siri, ChatGPT (current)	Hypothetical future
Existence	Exists today	Still in research